Demo Abstract: Automating WSN experiments and simulations

Rémy Léone*† Jérémie Leguay* Paolo Medagliani* Claude Chaudet†
* Thales Communications & Security – Gennevilliers, France
† Institut Mines-Télécom / Télécom ParisTech / CNRS LTCI UMR 5141 – Paris, France

***Abstract*—In wireless sensor networks (WSNs), as in every other discipline, people willing to evaluate the performance of an application or a protocol rely on modeling, simulation or experimentation. Simulations and models produce results for large-scale networks in a reasonable time, but trade representation accuracy for speed and hence ignore many physical and system effects, such as interference from the outside world or race conditions inside the nodes. Experimentation provides more representative and precise results, but is limited to small networks. Besides, they require more effort to be deployed and to collect results. These approaches are therefore complementary and should all be involved in the evaluation, which is seldom true, as it requires duplicating the deployment and data collection processes.**

In this demonstration, we present MakeSense, a framework that simplifies these tasks for both simulation and real experiments environments by creating a whole experimentation chain from a single JSON description file. By using MakeSense, it is possible to organize the compilation, to orchestrate the firmware deployment, to efficiently collect results and to plot statistics. We illustrate the ease of use and efficiency of the complete MakeSense workflow over a simple RPL-UDP deployment scenario evaluated with the Cooja simulator and the FIT IoT-Lab open testbed.

I. Introduction

Internet of Things (IoT) is growing interest from both industrial and scientific communities. Potentially, 50 billions of smart objects will be connected in 2020. This means that a significant effort of design thinking must be carried out while developing solutions for constrained Wireless Sensor Networks (WSNs). In particular, one of the major concerns of nowadays applications is how existing protocols scale for large networks.

The design and the validation of efficient protocols can be achieved via both simulators and realistic testbeds. However, both approaches present drawbacks. With simulators it is possible to carry out performance evaluation for a very large number of nodes in a reasonable time frame. On the other side, simulators like Cooja or Tossim, designed for constrained IoT networks, make several assumptions on physical and system aspects for tractability. Real testbeds such as FIT IoT-Lab[2] allow very accurate performance analysis. However, scaling experiments to a large number of devices often require a great effort for firmware deployment and statistic collection.

MakeSense[1] unveils the problems of validating IoT applications by offering a whole chain of experimentation. Through *a single JSON configuration file*, it is possible to compile, deploy, and run an experiment in FIT IoT-Lab or in Cooja. MakeSense also allows analyzing results by collecting and parsing the log files, to plot the results, and to generate HTML reports. Each of these steps can be executed separately or take part in a batch sequence.

This demonstration illustrates the ease of use and capacity of MakeSense over a simple application. The whole description of the experimental setup fits in a single JSON file. We present the syntax of this file and apply slight modification to use alternatively the Cooja simulator and the FIT IoT-Lab open experimental platform. The JSON file also contains directives on which parameters to measure, such as the energy consumption or routing overhead, on how to collect data and on how to present results.

II. MakeSense

MakeSense was introduced in a short paper [1] as a tool to easily organize, run and share WSN simulations. It is distributed under the Apache license through GitHub[1]. The version demonstrated here adds features and supports execution of the experiment using ContikiOS on the FIT IoT-Lab [2].

In MakeSense, the experimenter defines his workflow by specifying a sequence of steps, as illustrated on Figure 1. A single JSON configuration file orchestrates the whole chain, from scenario specification to graphs generation. The experimenter usually first invokes the `make` step to produce the binary files from his source code by typing the command `fab make:dummy` which compiles the source code for an experiment called *dummy*. It can then deploy these binaries on the remote platform using `fab push_iotlab:dummy` or in the Cooja simulator. The remote deployment details are configured within the JSON file that should contain a `nodes` directive that specifies how binaries are named and where they shall be uploaded.

Binaries names and destinations can be specified explicitly or by defining a function, whose syntax use variables and loops, as illustrated in listing 1. This example tells MakeSense to write in the `iotlab.json` JSON file lines to produce 43 firmware binaries named `dummy_1.iotlab-m3`, etc. from a template names `dummy_iotlab.json` and to store these binary files in a directory specified in the `path` variable. These binaries go on the Grenoble IoT Lab platform named `m3-1.grenoble.iot-lab.info`, etc. This function is then invoked in the MakeSense workflow by `fab push_iotlab:dummy`. Using templates ease firmwares management while improving its safety and transparency.

[1] http://github.com/sieben/makesense
[2] https://www.iot-lab.info

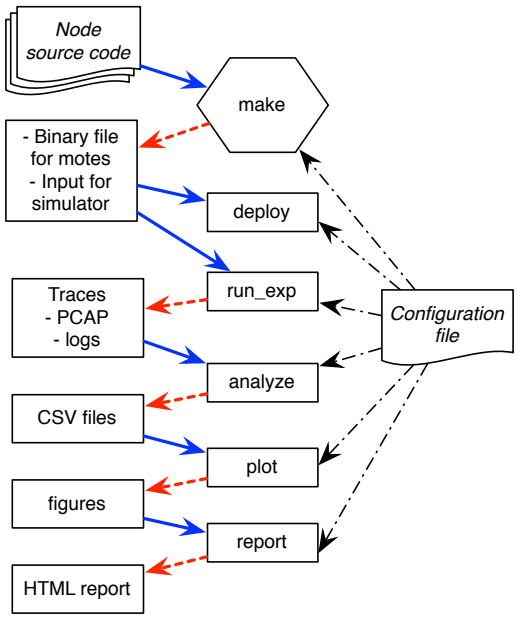

Fig. 1: MakeSense workflow

Listing 1: Configuration file excerpt

```
 def FUNCTION(name):
 [...]
# Location of a JSON file template to include
config_template =
       TEMPLATE_ENV.get_template("dummy_iotlab.json")

# Creation of the array that specifies and associates
# nodes names and with firmware binaries paths
 res = [
     {"nodes": ["m3-%d.grenoble.iot-lab.info" % num],
    "firmware_path": pj(path, "dummy_%d.iotlab-m3" % num)
    } for num in range(1, 43)]

# Dump the array in the JSON file
 with open(pj(path, "iotlab.json"), "w") as f:
     f.write(json.dumps(res))
 [...]
```

The experimenter can then execute the experiment with the `run_exp` step. Once the execution is finished, traces and log files are retrieved and analyzed with the `analyze` step, producing CSV files containing the desired metrics. The `plot` step creates graphs that can be inserted automatically in an HTML report with the `report` step.

III. Demonstration scenario overview

The demonstration consists in configuring, building and running an UDP application over a network running the RPL routing protocol. N nodes, ($N \in [10; 40]$) send UDP packets to a single destination, root of the RPL tree during 1 hour. We collect trace files that log the messages that nodes exchange and the RPL periodical control messages.

We show how to run the experiment on the Cooja simulator and on the ARM M3 node made available in the FIT IoT-lab Grenoble testbed. This platform allows monitoring the nodes

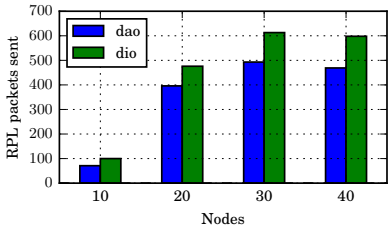

Fig. 2: RPL overhead as a function of the number of nodes.

real energy consumption with an embedded solution whose output can be collected by MakeSense. During experiments execution, different messages are aggregated and analyzed using a MakeSense step.

To illustrate the output of MakeSense, we measure the overhead due to the routing protocol, with the same syntax that was utilized in [1]. Once results are collected they are parsed and analyzed automatically, producing the graph shown in Figure 2, where the network-aggregated RPL overhead is shown as a function of the number of nodes in the network. The step generating the figure is exactly the same as the one used to parse and plot the results of a simulation run.

The last step, *report*, produces an HTML file that presents the results and the JSON configuration file, which can easily be shared. Visitors or collaborators have all the necessary information to re-run the experiment and compare results.

IV. Conclusion

This demonstration shows how to use MakeSense for automating the experimental process, not only in simulation with Cooja, but also using an open remote experimental platform. The heart of the demonstration consists in examining the configuration file and modifying it to change the simulation setup and its output, evaluating a simple RPL-UDP application deployed over FIT-IoT Lab.

MakeSense is currently configured to interact with Cooja and FIT IoT Lab but can easily adapt to other experimental platforms, depending on the existence of certain functionalities such as remote upload of the firmware binaries, remote access and data collection capabilities.

MakeSense was built using Python and uses a few libraries. It can easily be extended and future works will investigate, libraries such as iPython to provide dynamic interaction. iPython allows also the embedding of images, text and functions inside a single notebook along the variables to ease the sharing of an experiment.

V. Acknowledgment

This work is supported by the French National Research Agency (ANR) under grant reference IRIS ANR-11-INFR-0016.

References

[1] Rémy Léone, Jérémie Leguay, Paolo Medagliani, and Claude Chaudet, "Makesense: Managing reproducible wsns experiments," *Fifth Workshop on Real-World Wireless Sensor Networks*, 2013.